INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

New Low Cost and Undedicated Genetic Operators

Blaise MADELINE

N° 4573

Septembre 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4573--FR+ENG

New Low Cost and Undedicated Genetic Operators

Blaise MADELINE*

Thème 2 — Génie logiciel
et calcul symbolique
Projet Coprin

Rapport de recherche n° 4573 — Septembre 2002 — 14 pages

Abstract: The mutation and cross-over operators are, with selection, the foundation of genetic algorithms. We show in this paper, some possibilities offered by these operators. Having explained the specificity of the most known operators (1-point, p-point and uniform cross-over, classical and deterministic mutation) we introduce new crossover and mutation operators with a low cost in term of execution time. These operators were designed for Constraint Satisfaction Problem solving, but can be useful in other fields.We also introduce a new diversification operator for graph coloring.

Key-words: genetic algorithm, crossover, mutation, diversification operator. graph coloring

* INRIA/CERMICS/I3S

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Nouveaux opérateurs génétiques rapides et non dédiés

Résumé : Les opérateurs de mutation et de croisement sont, avec la sélection, les fondements des algorithmes génétiques. Nous montrons dans ce rapport, quelques possibilités offertes par ces opérateurs. Après avoir expliqué les spécificités des opérateurs les plus connus (croisement 1-point, p-point et uniforme, mutation classique et déterministe) nous introduisons de nouveaux opérateurs de croisement et de mutation qui ont un faible coût en temps d'exécution. Ces opérateurs ont été développés pour la résolution de problèmes de satisfaction de contraintes, mais peuvent être utile dans d'autres domaines. Nous introduisons aussi un nouvel opérateur de diversification pour le coloriage de graphe.

Mots-clés : algorithme génétique, croisement, mutation, opérateur de diversification, coloriage de graphe

1 Introduction

Genetic algorithm(GA) are widely used to solve a large area of hard optimization problems, and many new genetic operators have been presented as dedicated to solve a given problem or to implement new ideas. The design of a specialized operator causes some side effects: the more it is specialized the more it has a high cost in term of execution time. The other solutions are to design a real hybrid algorithm, for example with Tabu search [9] or simulated annealing [11], but it is not our goal here. The aim of our research is to find low cost operators, efficient on a large number of problems, especially in graph coloring and Constraint Satisfaction Problems(CSP). To apply genetic algorithm to CSPs is easy to do [6]. The broadly used model is the following :

- A chromosome in a genetic algorithm codes a configuration of instantiated variables.
- A gene of a chromosome represents a variable of the CSP.
- The values of the domain of this variable are represented by the allelic values of the corresponding gene.
- The evaluation function counts the number of violated constraints.
- The mutation operator changes the gene value by another value from the variable domain.

For our experiments, we used a GA Engine called AgCSP [4], which is dedicated to solve discrete CSPs. It works like a standard GA, but the chromosomes are integer coded instead of being binary coded. It is easier to treat problems as graph coloring or random CSP using this coding when the domains of the variables are finite. The new operators which we will present are designed for Constraint Satisfaction Problems(CSPs) but they can be used for many areas of optimization problems, and they have very low cost in term of execution time.

In section 2 we will be interested on crossover, we will show the limitations and the advantages of the known operators (1-point, p-point, uniform), before introducing two new crossover operators. In section 3, we present the classical mutation operators, some known deterministic mutation operators, and introduce new adaptive and deterministic mutation operators. And then in section 4 we propose a diversification operator for graph coloring, before to conclude.

2 CrossOver operators and CSP

In this section we discuss about the efficiency of classical 1-point and p-point crossover for solving CSPs. We show that because of the constraint graph structure, these crossovers are not well suited. We will present the well known uniform crossover[17], and will introduce a new operator with a different dynamic than uniform crossover. We will also present an adaptive crossover which is fully described in section 3.3.

2.1 Classical 1 point and p-point crossover

1-point and p-point crossover operators have been directly inspired from nature where the genetic code has a reading direction, and two genes are all the more linked as they are close. Therefore using a one point crossover in a GA seemed to be the best idea. But for a CSP, the linear representation of the CSP variables in a chromosome does not reflect the structure of the constraint graph. The co-dependency between two variables cannot be represented by two close genes. With the 1-point crossover, many constraint graph partitions are omitted, therefore we can miss some good solutions because the convergence follows just one direction. Indeed, in a given graph with n nodes, there is 2^n possible partitions in two subgraphs, and the 1-point crossover just proposes n partitions among all possible, however the aim of crossing is to exploit the maximum of possible reconfigurations. To avoid this problem the p-point crossover has been designed, but with a fixed number of points p we have only: $\binom{n}{p}$ partitions.

These crossovers are very interesting for some optimization problems, but are not satisfactory for CSPs and graph coloring problems.

2.2 Uniform crossover

Solving a graph coloring problem can be formulated as follow: given a graph $G = (V, E)$ and an integer k, finding a partition of V in k classes C_c such that $\forall i \in C_c, \forall j \in C_c$, $(i, j) \notin E$. The uniform crossover [17] has been designed to take into account the specificity of graph partitioning, and to have a better crossover for problems where the co-dependency is not represented by close genes. With this operator each locus has the same chance to be chosen. Each parent's gene has a probability of $\frac{1}{2}$ to be represented in the child chromosome.

With this crossover, we have 2^n possible different crossings, and a probability of $\frac{1}{2^n}$ for each, here from the name: uniform crossover. But the number of chosen points for crossing is not uniform : a 1 point crossover has $\frac{n}{2^n}$ chance to represented and a p points has $\frac{\binom{n}{p}}{2^n}$ chance. The probability to have a p points crossing follows a curve traced on figure 1. The crossings with the minimum number of points, and those with a maximum number of points are badly represented, and those with nearly $\frac{n}{2}$ points are highly represented. To take into account the specificity of some problems, we wanted to have a uniform probability for the number of crossing points, so we defined a new operator called p-rand-point crossover.

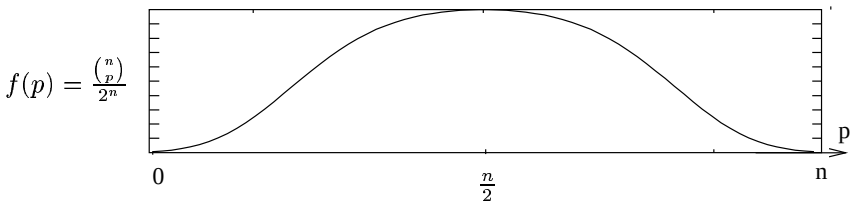

Figure 1: Probability to have p crossing points

2.3 p-rand point crossover

We want a crossover with a uniform repartition of the number of crossing points. But we do not want to compute a complex probability which will burden the algorithm complexity. We choose a simple algorithm in three steps which has the following behavior:

- Choose randomly the number p of crossing points between 0 and n.
- Choose randomly the positions of the crossing points, between the $\binom{n}{p}$ different possible positions.
- Execute the crossing like a classical p-point crossover.

Let us calculate the number of possible different crossings:

- We have n possible number of crossing points .
- For p chosen points, we have $\binom{n}{p}$ possible positions.

Finally, we have $\sum_{p=0}^{p=n} \binom{n}{p} = 2^n$ different possible crossings. Like for uniform crossover, we obtain all the possible graph partitions. We have a uniform probability to have a certain number p of crossing points, but for one particular crossing with p fixed points, its probability is: $\frac{1}{n} \times \frac{1}{\binom{n}{p}}$ which follows a curve traced on figure 2.

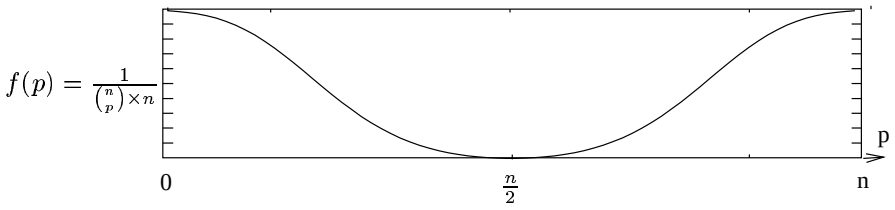

Figure 2: Probability to have a given configuration of crossing with p points

This operator has some advantages, we do not suppose that the number of crossing points is better if it is near $\frac{n}{2}$. We assume that it is interesting for the operator to not always break the same dependency, particularly for problems like the n-queen coloring problem (NQCP). In the NQCP the goal is to color all the squares of a chess board such that two squares cannot be on the same color if they are on the same column, the same line or the same diagonal. To code this problem as a chromosome, each line of the chess board is represented contiguously. In fact, preserving some co-dependency between the squares of a line can be very useful for the search. This crossover is also interesting when a greedy initialization is used. A simple way for doing a greedy initialization is to choose a first gene randomly, and initialize the next genes by choosing randomly between the best values. In this case, breaking the satisfied constraints is not interesting, and our crossover better preserves this dependency than uniform crossover.

2.4 adaptive crossover

The parameter control is one of the difficulties arising from GAs [5], and adaptive methods are one of the solutions to ease the tuning. Every operator can include an adaptive behavior: In [7] Eiben and van der Hauw have proposed an adaptive evaluation function with stepwise adaptive weights (SAW) to solve graph coloring problems: the penalty associated to a variable is increased when some constraints are violated. Different adaptive mutation operators were also proposed (see section 3), but in this section, we are interested on adaptive crossover operators. A constraint dynamic adapting crossover is proposed in [15] by Riff-Rojas. With this crossover the child inherits its genes using a greedy procedure, which analyses each constraint according a dynamic priority. This dynamic priority takes into account the network structure and the value of the parents. The first constraint to be analyzed is the hardest to be satisfied. Smith and Fogarty propose a multi-parent recombination [16], which takes into account the linked genes, defined dynamically by a flag as a block of genes over certain loci.

There are many other ways to design adaptive operators. Nevertheless methods for changing parameters during the run can be classified in three categories: deterministic, exogenous (i.e. adaptive) and endogenous (i.e. self-adaptive) methods [5, 1]. The deterministic methods control the parameters by a strategy chosen before the run, and do not use any feedback from the run. The exogenous methods use the informations given by the run to change the parameter values, and the endogenous methods code the parameter control in the genes of the chromosome itself. We consider that deterministic methods are more interesting for static problems, endogenous methods are well designed for dynamic problems, and exogenous can be used on the both problem types. We are interested here by the exogenous methods. We defined an adaptive crossover, in which the rate is decreased when the population is too homogeneous, and increased when the population is too heterogeneous. This operator is fully explained in section 3.3 with the adaptive mutation which has the same behavior.

3 Mutation operators

Mutation operators are the diversification operators. Without diversification, a GA converges to a local optimum and cannot escape from it. But with a classical GA, it is very difficult to find the best mutation rate, in fact $\frac{1}{n}$ seems to be a good value in general case [12]. The idea here is to find new mutation operators, with a low cost in execution time, and not specialized for a given problem. We first talk about the classical mutation operator, introduce a new deterministic mutation operator and a new adaptive mutation collectively explained with the adaptive crossover introduced before.

3.1 Classical allelic mutation

First we have to distinguish the allelic mutation from the chromosomic mutation. With the chromosomic mutation each chromosome has a probability to be muted, while with the allelic mutation each gene of a chromosome has a probability to be muted. Then with allelic mutation, each chromosome has on average one gene muted, with a mutation rate set to $\frac{1}{n}$. Although the rate of $\frac{1}{n}$ generally

seems to be a good mutation rate, it becomes less interesting when the algorithm is in an attractive basin of a local optimum or deceiver site. In this paper we always use allelic mutation.

3.2 Deterministic mutation

It is known that a high rate for mutation increases the diversity and a low rate helps the convergence, and it is difficult to find the good compromise. One way is to decrease the mutation rate during the run. The first one has been proposed by Fogarty in [8]. Many deterministically decreasing schemes for mutation rate have been proposed further in literature, the idea came from simulated annealing, which less accepts bad value in the neighborhood at the end of the run than at the beginning. A decreasing mutation rate for counting-ones function has been proposed by Hesser and Manner in [10], which takes into account the generation count, and the population size:

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \times \frac{exp(\frac{-\gamma t}{2})}{\lambda\sqrt{n}} \tag{1}$$

where α, β, γ are constant, λ the population size, n the number of genes in a chromosome, and t the generation counter. Another idea is to decrease the mutation rate as a function of the distance to the optimum proposed by Bäck in [2]:

$$p_m(f(\vec{x})) \approx \frac{1}{2(f(\vec{x})+1)-n} \tag{2}$$

Bäck also presents with Schütz [3] a decreasing mutation rate from 0.5 to $\frac{1}{n}$ when the number of evaluations is known before the run:

$$p_m(t) = \left(2 + \frac{n-2}{T} \times t\right)^{-1} \tag{3}$$

n is the chromosome's length, T the maximum number of evaluations and t the current number of evaluations.

These operators are very interesting but their limitations are due to the blocking of the GA in a local optimum near the end of the run. It is then impossible to go out. When the GA is blocked in a local optimum, it would be interesting to increase the mutation rate. But it is difficult to locate such a blocking in a local optimum during the run. Furthermore, the mutation rate must be decreased when the algorithm escaped from this local optimum. A solution is to increase slowly the mutation rate during the run. The hint came from the random initialization that let a good diversity in the population. The diversity decreases during the run due to selection pressure. In fact, increasing the mutation rate may preserve a good diversity, but this makes too much exploration and does not really work, furthermore the regulation is quite impossible.

We think that combining increase and decrease schemes can give an efficient operator, so we propose a mutation operator in which the rate follows a sinus curve. This allows the algorithm to go out the local optima, and conserve a good convergence. The mutation rate is defined as follows:

$$p_m(t) = P_M + \left(sin\left(\frac{t}{\gamma.\pi}\right) \times \alpha\right) \tag{4}$$

where α, γ are constants, P_M the default mutation rate (generally $\frac{1}{n}$), n the chromosome's length and t the generation counter.

- the period of the sinus is determined by the constant γ (given in number of generations).
- the amplitude of the sinus is determined by the constant α and $\alpha < P_M$.

During the run, the GA always alternates between exploration and exploitation, actually the convergence takes more time. Nevertheless less runs stay blocked in a local optimum and better values are sometime found after a period of high exploration (i.e. when the mutation rate is high). An example is given in figure 3. This operator has a really good behavior when the problem has many local

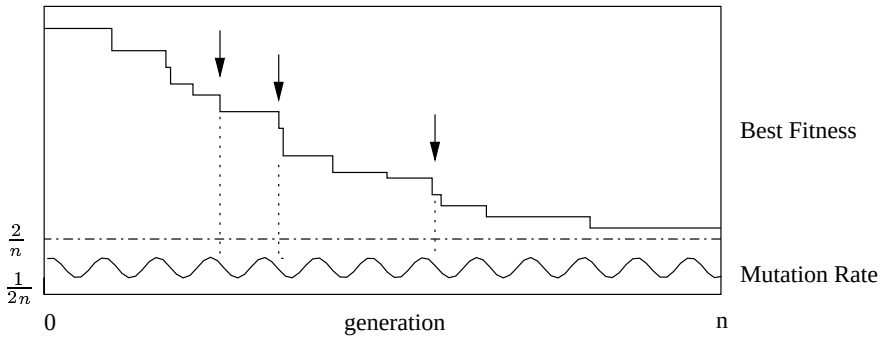

Figure 3: Sinus mutation and best fitness

optima for which we can easily go out with a higher mutation rate. We also integrate an amortized sinus which allows to explore in a larger neighborhood at the beginning of the run than at the end. This idea combines the decreasing rate idea and the first implementation of our operator:

$$p_m(t) = P_M + \left(sin\left(\frac{t}{\gamma.\pi}\right) \times \frac{\alpha}{t} \right) \tag{5}$$

which has the same scheme than sinus mutation, we have just add a decreasing factor with the generation counter. With these two operators, it is strongly advised to use elitism, because the high mutation rate can make lose good solutions.

3.3 Adaptive allelic mutation

With the same idea as the adaptive crossover introduced above (see section 2.4), we propose an adaptive mutation operator. The principle is to increase the mutation rate when the population is too homogeneous, and decrease it when the population is too heterogeneous. We need a diversity measurement function, but we wanted one very simple to compute. In our GA engine, the fitness function counts the number of violated constraints. That gives us two important informations:

- The best wished fitness is 0, and we directly have a measurement of the distance from the best found to the best wished in number of violated constraints.
- We have a measurement between the best found and the average fitness in term of violated constraints.

We assume that the ratio between the best fitness and the average fitness can be a good diversity measurement function simple to achieve and of low cost because it uses already computed values. Our diversity measure DM is defined as follows:

$$DM = 1 - \frac{BestFitness}{AverageFitness} \tag{6}$$

These adaptive operators have parameters: we have to define the bounds for the mutation rate and crossover rate, i.e. the minimum rate ($p_{m_{min}}$, $p_{c_{min}}$) and the maximum rate ($p_{m_{max}}$, $p_{c_{max}}$) accepted for the run. For example we cannot accept a mutation rate set to 0.6 or a crossover rate set to 0.01. We also have two bounds for the diversity measurement. The rates are adapted only when the diversity measurement DM is greater (resp. lower) than the bound DM_{max} (resp. DM_{min}). Finally we define a increase/decrease factor. The algorithms for adapting the mutation rate and the crossover rate are:

Mutation rate :	CrossOver rate:
compute DM;	compute DM;
if DM $< DM_{min}$ and $p_m < p_{m_{max}}$	**if** DM $< DM_{min}$ and $p_c > p_{c_{min}}$
then increase p_m;	**then** decrease p_c;
if DM $> DM_{max}$ and $p_m > p_{m_{min}}$	**if** DM $> DM_{max}$ and $p_c < p_{c_{max}}$
then decrease p_m;	**then** increase p_c;
else do nothing;	**else** do nothing;

The parameters are defined as follows:

- $p_{m_{min}}$ (minimum rate) is the lowest mutation rate permitted (generally $\frac{1}{10\times n}$). Under this rate, no real exploration is possible and $p_{c_{min}}$ (minimum rate) is the lowest crossover rate permitted (generally 0.50). Under this rate, no real exploitation is possible.
- $p_{m_{max}}$ (maximum rate) is the highest mutation rate permitted (generally $\frac{10}{n}$).But it can be higher for difficult problems and $p_{c_{max}}$ (maximum rate) is the highest crossover rate permitted (generally 0.90).
- DM_{min} (Minimum bound) is the lowest diversity measure allowed. (between 0.05 and 0.10 seems to be the best value)
- DM_{max} (Maximum bound) is the highest diversity measure allowed (between 0.10 and 0.15 seems to be the best value)
- the increasing or decreasing factor (the best value seems to be between 1.1 and 1.5)

Remark that the three last parameters have the same tuning both for adaptive crossover and adaptive mutation. The value of the increase/decrease factor is a reactivity factor: higher it is, quicker is the reaction on the rate. Tests show that these operators have a good robustness compared to classical mutation and crossover. Furthermore their parameters do not require fine tunings. Using these operators gives a good result for the GA convergence in most of cases, but does not outperform the best run with the best tunings of classical operators. But no long time is spent in parameter tuning: these parameters are easy to control and within a few tests, a good parameterization is found. It is really interesting to use both adaptive mutation and crossover, with exactly the same tuning for DM_{min} and DM_{max}. With this tuning, the crossover rate decreases (resp. increases) when the mutation rate increases (resp. decreases), and the adaptive process is done collectively, but it is also possible to use different tuning for DM_{min} , DM_{max} and the factor of increase or decrease. After testing, adaptive mutation appears to be more efficient than adaptive crossover. Indeed, the crossing has a much slower dynamics than mutation, and adaptive crossover does not bring notorious improvement.

4 Diversification operator

Avoiding early convergence is one of the main care for GA, and there are many methods therefore. Those methods try to reintroduce some diversity in the population or to conserve a good diversity. Many solutions are offered to do it: introducing new randomized individuals, changing mutation rate, doing an hybrid search, using a steady state algorithm or a co-evolutionary algorithm for example.

In graph coloring, a high number of potential solutions have the same evaluation with very different gene valuations. Two solutions can be genotypically different but phenotypically identical. For example on Fig.4, the two graphs have exactly the same number of violated constraints, but are really different. No node has the same color. And their corresponding chromosomes are different too.

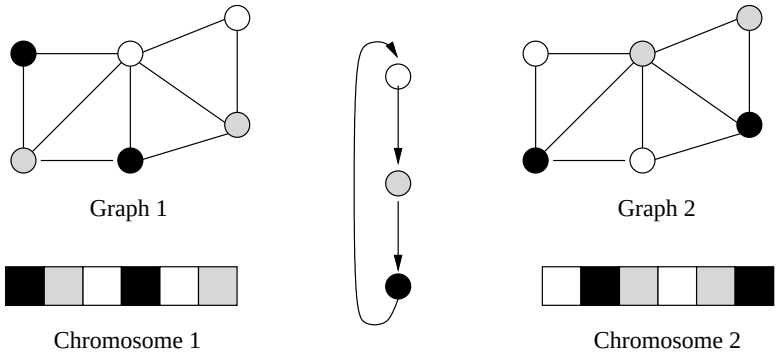

Figure 4: Two solutions with different genotypes and same phenotype, and the swap operation from the first to the second

Furthermore it is very easy to find the second instantiation from the first: it is a swap from the domain. This idea is easily transposable to a genetic operator. Our GA engine uses an integer coding for the gene. This operator chooses a value randomly from the domain, and add this value to all the genes of the chromosome modulo the domain size. Before the run we have to choose the percentage of the population which will be swapped. The best results have been obtained with a swap rate set to 10 percents of the population size per generation. If the swap rate is too high the population is too heterogeneous and the exploitation is very hard.

As mentioned above, potential solutions for a CSP can have many different genotypes and very similar fitness. For this reason, another interesting idea is to keep more than one of the best individuals found during the run. Having an elitism rate set to 0.10 can conserve genotypically different best solutions, and give better results. This is due to exploitation done in different directions. The GA can use different current best solutions to find a better one.

5 Conclusion

In this paper, we have presented a new crossover operator for CSPs, efficient on certain graph coloring problems. It will be interesting to test it on other problems where the structure let some cliques be contiguous in the chromosome, like for the NQCP. It will be also interesting to transform it with adaptive parameters, especially for the number of chosen points by adapting the probability to have a certain number of crossover points. We propose a sinus mutation which combines the advantages of the decreasing mutation and increasing mutation, and allows the GA to escape from local optima while preserving the convergence. Although very promising results were founded, it will be interesting to test and compare to other mutation operators on difficult instances. We present adaptive mutation and crossover, which let us lose much fewer time in parameters tuning, and give better results on average than classical operators. This can be very useful when a good solution is needed quickly. We also present a diversification operator dedicated to graph coloring using the difference between phenotype and genotype. We think these operators can be efficient on other areas where the GAs are applied but new experiments are necessary to improve this.

References

[1] Peter J. Angeline. Adaptive and self-adaptive evolutionary computations. In M. Palaniswami, Y. Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, Piscataway, NJ, 1995.

[2] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, pages 85–94, Amsterdam, 1992. Elsevier.

[3] Thomas Bäck and Martin Schütz. Intelligent mutation rate control in canonical genetic algorithms. In *International Syposium on Methodologies for Intelligent Systems*, pages 158–167, 1996.

[4] F. Didierjean. AgCSP, une bibliothèque de classes C++ pour le développement d'opérateurs génétiques pour CSP, 2002. PhD in preparation.

[5] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in Evolutionary Algorithms. *IEEE Transaction on Evolutionnary Computation*, 3(2):124, July 1999.

[6] A. E. Eiben, P.-E. Raue, and Zsofia Ruttkay. GA-easy and GA-hard constraint satisfaction problems. In Manfred Meyer, editor, *Proc. of European Conferance on Artificial Intelligence, Workshop on Constraint Processing*, pages 267–283, Amsterdam, 1994.

[7] A. E. Eiben and J. K. van der Hauw. Adaptive penalties for evolutionary graph coloring. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution – Third European Conference*, pages 95–106, Berlin, 1997. Springer.

[8] Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms*,, pages 104–109, 1989.

[9] P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[10] Jurgen Hesser and Reinhard Manner. Towards an optimal mutation probability for genetic algorithms. In *Parallel Problem Solving from Nature*, pages 23–32, 1990.

[11] F. T. Lin, C. Y. Kao, and C. C. Hsu. Applying the genetic approach to simulated annealing in solving some NP-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1752–1767, - 1993.

[12] H. Mühlenbein. How genetic algorithms really work: I. mutation and hill-climbing. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 15–25, Amsterdam. Elsevier., 1992.

[13] Nicholas J. Radcliffe and Felicity A. W. George. A study in set recombination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 23–30. Morgan Kaufmann Publishers, 1993.

[14] Nicholas J. Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In *Computer Science Today*, pages 275–291. 1995.

[15] M.-C. Riff-Rojas. A network-based adaptive evolutionary algorithm for constraint satisfaction problems. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pages 269–283. Kluwer Academic Publishers, 1998.

[16] Jim E. Smith and Terence C. Fogarty. Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm. In H. Voigt, W. Ebeling, and I. Rechenberg, editors, *Parallel Problem Solving from Nature IV*, pages 441–450, Berlin, 1996. Springer.

[17] Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. D. Shaeffer, editor, *Proceeding of Third International Conference on Genetic Algorithms and Their Applications*, pages 2–9, 1989.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399